Amendment to the Specification:

The Paragraph beginning at Page 57, line 8, is to be amended as follows:

DRAM or direct CPU writes, in the order defined in ~~Table~~ Table 12.

The Paragraph beginning at Page 57, lines 34 - 35, is to be amended as follows:

2) Shutdown the PEP blocks by de-asserting their Go registers in the suggested order in ~~Table~~ Table 13. This will set the PEP Unit state-machines to their startup states.

The Paragraph beginning at Page 86, lines 2 - 7, is to be amended as follows:

The CPU Subsystem Bus Interface block performs simple address decoding to select a peripheral and multiplexing of the returned signals from the various peripheral blocks. The base addresses used for the decode operation are defined in ~~Table~~ Table 23. Note that access to the MMU configuration registers are handled by the MMU Control Block rather than the CPU Subsystem Bus Interface block. The CPU Subsystem Bus Interface block operation is described by the following pseudocode:

The Paragraph beginning at Page 189, lines 19 - 22, is to be amended as follows:

The BLDC controller logic is identical for both instances, only the input connections are different. The logic implements the truth table shown in ~~Table~~ Table 83. The six *q* outputs are combinationally based on the *direction*, *ha*, *hb*, *hc* and *pwm* inputs. The direction input has 2 possible sources selected by the mode, the pseudocode is as follows

The Paragraph beginning at Page 243, lines 3 - 8, is to be amended as follows:

Since DRAM read requestors, except for the CPU, are connected to the DIU via a 64-bit data bus each 256-bit DRAM access requires 4 *pclk* cycles to transfer the read data over the shared read bus. The timeslot rotation period for 64 timeslots each of 4 *pclk* cycles is 256 *pclk* cycles or 1.6 µs, assuming *pclk* is 160 MHz. Each timeslot represents a 256-bit access every 256 *pclk* cycles or 1 bit/cycle. This is the granularity of the majority of DIU requestors bandwidth requirements in ~~Table~~ Table 111.

The Paragraph beginning at Page 246, lines 12 - 14, is to be amended as follows:

The CPU is not included in the list of SoPEC DIU requesters, ~~Table~~ Table 111, for the main timeslot allocations. The CPU cannot therefore be allocated main timeslots. It relies on pre-accesses in advance of such slots as the sole method for DRAM transfers.

The Paragraph beginning at Page 247, lines 10 - 11, is to be amended as follows:

Refresh is not included in the list of SoPEC DIU requesters, ~~Table~~ Table 111, for the main timeslot allocations. Timeslots cannot therefore be allocated to refresh.

The Paragraph beginning at Page 254, line 1, is to be amended as follows:

Table 121 shows an allocation of main timeslots based on the peak bandwidths of ~~Table~~ Table 109.

The Paragraph beginning at Page 254, line 8 - 14, is to be amended as follows:

Program the *MainTimeslot* configuration register (~~Table~~ Table 121) for peak required bandwidths of SoPEC Units according to the scale factor.

Program the read round-robin allocation to share unused read slots. Allocate PCU, DNC, HCU and TFS to level 1 read round-robin.

- Assume scale-factor of 6 and peak bandwidths from ~~Table~~ Table 109.
- Assign all DIU requestors except TE(TFS) and HCU to multiples of 1 timeslot, as indicated in ~~Table~~ Table 121, where each timeslot is 1 bit/cycle. This requires 33 timeslots.

The Paragraph beginning at Page 255, line 8 - 13, is to be amended as follows:

Program the *MainTimeslot* configuration register (~~Table~~ 121) for peak required bandwidths of SoPEC Units according to the scale factor. Program the read round-robin allocation to share unused read slots. Allocate PCU, DNC, HCU and TFS to level 1 read round-robin.

- Assume scale-factor of 4 and peak bandwidths from ~~Table~~ Table 109.

- Assign all DIU requestors except TE(TFS) and HCU multiples of 1 timeslot, as indicated in ~~Table~~ Table 121, where each timeslot is 1 bit/cycle. This requires 38 timeslots.

The Paragraph beginning at Page 257, line 21, is to be amended as follows:

~~Table~~ Table 113 estimated the CPU read latency as 6 cycles.

The Paragraph beginning at Page 261, line 36 - 40, is to be amended as follows:

During a random read, the read data is returned, on *dcu_dau_rdata*, after time T_{acc}, the random access time, which varies between 3 and 8 ns (see ~~Table~~ Table 128). To avoid any metastability issues the read data must be captured by a flip-flop which is enabled 2 *pclk* cycles or 12.5 ns after the DRAM access has been started. The DCU generates the enable signal *dcu_dau_rvalid* to capture *dcu_dau_rdata.*

The Paragraph beginning at Page 277, line 3 - 7, is to be amended as follows:

b. The time between a DIU requester requesting an access and completing the access.

This information can be obtained by observing the signals in the *DIUPerformance* debug register at *DIU_Base+0x308* described in Table 135. The encoding for *read_sel* and *write_sel* is described in ~~Table~~ Table 135. The data collected from *DIUPerformance* can be post-processed to count the number of cycles between a unit requesting DIU access and the access being completed.

The Paragraph beginning at Page 283, line 19 - 22, is to be amended as follows:

When the Command Multiplexor initiates arbitration by asserting *re_arbitrate* to the Arbitration Logic sub-block, the arbitration winner is indicated by the *arb_sel[4:0]* and *dir_sel[1:0]* signals returned from the Arbitration Logic. The validity of these signals is indicated by *arb_gnt*. The encoding of *arb_sel[4:0]* is shown in ~~Table~~ Table 134.

The Paragraph beginning at Page 295, line 3 - 9, is to be amended as follows:

Arbitration is triggered by the signal *re_arbitrate* from the Command Multiplexor sub-block with the signal *arb_gnt* indicating that arbitration has occurred and the arbitration winner is indicated by *arb_sel[4:0]*. The encoding of *arb_sel[4:0]* is shown in ~~Table~~ Table

134. The signal *dir_sel[1:0]* indicates if the arbitration winner is a read, write or refresh. Arbitration should complete within one clock cycle so *arb_gnt* is normally asserted the clock cycle after *re_arbitrate* and stays high for 1 clock cycle. *arb_sel[4:0]* and *dir_sel[1:0]* remain persistent until arbitration occurs again. The arbitration timing is shown in Figure 119.

The Paragraph beginning at Page 298, line 38 - 41, is to be amended as follows:

The write lookahead pointer points two timeslots in advance of the current timeslot pointer. Therefore *re_arbitrate_wadv* causes the Arbitration Logic to perform an arbitration for non-CPU two timeslots in advance. As noted in ~~Table~~ Table 114, each timeslot lasts at least 3 cycles. Therefor *re_arbitrate_wadv* arbitrates at least 4 cycles in advance.

The Paragraph beginning at Page 299, line 4 - 8, is to be amended as follows:

Some accesses can be preceded by a CPU access as in ~~Table~~ Table 115. These CPU accesses are not allocated timeslots. If this is the case the timeslot will last 3 (CPU access) + 3 (non-CPU access) = 6 cycles. In that case, a second write lookahead pointer, the CPU pre-access write lookahead pointer, is selected which points only one timeslot in advance. *re_arbitrate_wadv* will still arbitrate 4 cycles in advance.

The Paragraph beginning at Page 301, line 28 - 29, is to be amended as follows:

CPU and refresh are not included in the timeslot allocations defined in the DAU configuration registers of ~~Table~~ Table 130.

The Paragraph beginning at Page 302, line 15 - 25, is to be amended as follows:

Unused write accesses are re-allocated according to the fixed priority scheme of ~~Table~~ Table 116. Unused read timeslots are re-allocated according to the two-level round-robin scheme described in Section 20.10.6.2.

A pointer points to the most recently re-allocated unit in each of the round-robin levels. If the unit immediately succedling the pointer is requesting, then this unit wins the arbitration and the pointer is advanced to reflect the new winner. If this is not the case, then the subsequent units (wrapping back eventually to the pointed unit) in the level 1 round-robin are examined. When a requesting unit is found this unit wins the arbitration and the

pointer is adjusted. If no unit is requesting then the pointer does not advance and the second level of round-robin is examined in a similar fashion.

In the following pseudo-code the bit indices are for the *ReadRoundRobinLevel* configuration register described in ~~Table~~ Table 118.

The Paragraph beginning at Page 308, line 17 - 30, is to be amended as follows:

The encoding of *arb_sel[4:0]* is given in ~~Table~~ Table 134. *dir_sel[1:0]=="01"* indicates that the operation is a read. The read command queue is shown in Figure 128.

The command queue could contain values of *arb_sel[4:0]* for 3 reads at a time.

- In the scenario of Figure 127 the command queue can contain 2 values of *arb_sel[4:0]* i.e. for the simultaneous CDU and CPU accesses.
- In the scenario of Figure 130, the command queue can contain 3 values of *arb_sel[4:0]* i.e. at the time of the second *dcu_dau_rvalid* pulse the command queue will contain an *arb_sel[4:0]* for the arbitration performed in that cycle, and the two previous *arb_sel[4:0]* values associated with the data for the first two *dcu_dau_rvalid* pulses, the data associated with the first *dcu_dau_rvalid* pulse not having been fully transfered over the shared read data bus.

The read command queue is specified as 4 deep so it is never expected to fill.

The top of the command queue is a signal *read_type[4:0]* which indicates the destination of the current read data. The encoding of *read_type[4:0]* is given in ~~Table~~ Table 134.

The Paragraph beginning at Page 313, line 28 - 29, is to be amended as follows:

The encoding of *arb_sel[4:0]* is given in ~~Table~~ Table 134. *dir_sel[1]==1* indicates that the operation is a write. *arb_sel[4:0]* is only written to the write command register if the write is a non-CPU write.

The Paragraph beginning at Page 326, line 1 - 3, is to be amended as follows:

When in the *DRAMAccess* state the commands are executed from the *cmd_fifo*. A command in the *cmd_fifo* consists of 64-bits (or which the FIFO holds 4). The decoding of the 64-bits to commands is given in ~~Table~~ Table 142. For each command the decode is

Page 263 is to be amended as follows:

			arbitration winner.
re_arbitrate_wadv	1	Out	Signal telling the arbitration logic to choose the next arbitration winner for non-CPU writes 2 timeslots in advance
Debug Outputs to CPU Configuration and Arbitration Logic Sub-block			
write_sel	5	Out	Signal indicating the SoPEC Unit for which the current write transaction is occurring. Encoding is described in ~~Table~~Table 134.
write_complete	1	Out	Signal indicating that write transaction to SoPEC Unit indicated by *write_sel* is complete.
Inputs from CPU Interface and Arbitration Logic sub-block			
arb_gnt	1	In	Signal lasting 1 cycle which indicates arbitration has occurred and *arb_sel* is valid.
arb_sel	5	In	Signal indicating which requesting SoPEC Unit has won arbitration. Encoding is described in ~~Table~~Table 134.
dir_sel	2	In	Signal indicating which sense of access associated with *arb_sel* 00: issue non-CPU write 01: read winner 10: write winner 11: refresh winner
Inputs from Read Write Multiplexor Sub-block			
write_data_valid	2	In	Signal indicating that valid write data is available for the current command. 00=not valid 01=CPU write data valid 10=non-CPU write data valid 11=both CPU and non-CPU write data valid
wdata	256	In	256-bit non-CPU write data
cpu_wdata	32	In	32-bit CPU write data
Outputs to Read Write Multiplexor Sub-block			
write_data_accept	2	Out	Signal indicating the Command Multiplexor has accepted the write data from the write multiplexor 00=not valid 01=accepts CPU write data 10=accepts non-CPU write data 11=not valid
Inputs from DCU			

dcu_dau_adv	1	In	Signal indicating to DAU to supply next command to DCU

The Paragraph beginning at Page 370, line 1 - 3, is to be amended as follows:

Since the CDU, LBD and TE all access the page band store, they share two registers that enable sequential memory accesses to the page band stores to be circular in nature. The CDU chapter lists these two registers. The register descriptions for the LBD are listed in ~~Table~~ Table 156.

The Paragraph beginning at Page 373, line 3 - 7, is to be amended as follows:

Once this edge has been detected, the delta will denote which of the vertical commands to use, refer to ~~Table~~ Table 152. The delta will denote where the changing element in the current line is relative to the changing element on the previous line, for a Vertical(2) the new changing element position in the current line will correspond to the two bits extra from changing element position in the previous line.

The Paragraph beginning at Page 382, line 8 - 10, is to be amended as follows:

At the start of each line *count* in the pseudo-code above is set to *XstartCount*. If there is no lead-in, *XstartCount* is set to 0 i.e. the first value of *count* in ~~Table~~ Table 162. If there is lead-in then *XstartCount* needs to be set to the appropriate value of *count* in the sequence above.

The Paragraph beginning at Page 410, lines 18 - 28, is to be amended as follows:

The TFS consists of *TagHeight* number of *tag line structures*, one for each 1600 dpi line in the tag's bounding box. Each tag line structure contains three contiguous tables, known as tables A, B, and C. Table A contains 384 2-bit entries, one entry for each of the maximum number of dots in a single line of a tag (see ~~Table~~ Table 170). The actual number of entries used should match the size of the bounding box for the tag in the dot dimension, but all 384 entries must be present. Table B contains 32 9-bit data addresses that refer to (in order of appearance) the data dots present in the particular line. All 32 entries must be present, even if fewer are used. Table C contains two 5-bit pointers into table B, and therefore comprises 10 bits. Padding of 214 bits is added. The total length of each tag line structure is therefore 5×256-bit DRAM words. Thus a TFS containing *TagHeight* tag

line structures requires a *TagHeight* * 160 bytes. The structure of a TFS is shown in Figure 184.

The Paragraph beginning at Page 414, lines 10 - 21, is to be amended as follows:

In order to support scaling in the Y direction the following modifications to the PEC1 TE are suggested to the Tag Data Interface, Tag Format Structure Interface and TE Top Level:

- for Tag Data Interface: program the configuration registers of ~~*Table*~~ *Table 176*, *firstTagLineHeight* and *tagMaxLine* with true value i.e. not multiplied up by the scale factor *YScale*. Within the Tag Data interface there are two counters, *countx* and *county* that have a direct bearing on the *rawTagDataAddr* generation. *countx* decrements as tags are read from DRAM. It is reset to *NumTags[RtdTagSense]* at start of each line of tags. *county* is decremented as each line of tags is completely read from DRAM i.e. *countx* = 0. Scaling may be performed by counting the number of times *countx* reaches zero and only decrementing *county* when this number reaches *YScale*. This will cause the TagData Interface to read each line of tag data *NumTags[RtdTagSense]* * *YScale* times.

The Paragraph beginning at Page 431, lines 31 - 35, is to be amended as follows:

Both TD and TFS storage in DRAM can wrap around the bandstore area. The bounds of the band store are described by inputs from the CDU shown in ~~Table~~ Table 174. The TD and TFS DRAM interfaces therefore support bandstore wrapping. If the TD or TFS DRAM interface increments an address it is checked to see if it matches the end of bandstore address. If so, then the address is mapped to the start of the bandstore.